

**Sprawozdania**  
**Wojciech Wnuk**  
**IO 7.15**

**Zaawansowane programowanie w Javie**  
**VII semestr**  
**Rok akademicki: 2023/24**

## **Spis treści**

|  |    |
|--|----|
| LABORATORIUM 1. STRUMIENIE I PLIKI.....                                      | 3  |
| LABORATORIUM 2. WYKORZYSTANIE PLIKÓW XML.....                                | 7  |
| LABORATORIUM 6. TWORZENIE APLIKACJI Z WYKORZYSTANIEM BIBLIOTEKI LOMBOK. .... | 14 |
| LABORATORIUM 3. TWORZENIE APLIKACJI SIECIOWYCH.....                          | 19 |
| LABORATORIUM 10. TWORZENIE APLIKACJI Z WYKORZYSTANIEM BIBLIOTEKI JSOUP.....  | 27 |
| LABORATORIUM 11. WYBRANE ZAGADNIENIA DOTYCZĄCE BEZPIECZEŃSTWA. ....          | 34 |

# LABORATORIUM 1. STRUMIENIE I PLIKI

## Zadanie 1.1. Operacje na strumieniach

```
Main.java x
1 package com.company;
2
3 import java.io.*;
4 import java.util.*;
5 import java.util.stream.*;
6
7 public class Main {
8     public static void main(String[] args) {
9
10         // Zadanie 1.1: Operacje na strumieniach
11
12         List<String> przedmioty = new ArrayList<>();
13         przedmioty.addAll(Arrays.asList("Integracja systemow", "Interakcja czlowiek-komputer",
14             "Programowanie aplikacji mobilnych na platforme Android",
15             "Programowanie aplikacji mobilnych na platforme iOS"));
16
17         Stream<String> strumienPrzedmiotow = przedmioty.stream();
18         List<Integer> listaOczen = strumienPrzedmiotow
19             .filter(przedmiot -> !przedmiot.contains("Zaaw"))
20             .map(przedmiot -> {
21                 return (int) (Math.random() * 4) + 2;
22             })
23             .collect(Collectors.toList());
24
25         listaOczen.forEach(ocena -> System.out.println("1.1: Ocena: " + ocena));
26
27         Map<Integer, Long> liczbaPowtorzen = listaOczen.stream()
28             .collect(Collectors.groupingBy(Integer::intValue, Collectors.counting()));
29
30         liczbaPowtorzen.forEach((ocena, liczba) -> {
31             if (liczba > 1) {
32                 System.out.println("Ocena " + ocena + " powtarza się " + liczba + " razy.");
33             }
34         });
35     }
```

## Zadanie 1.2. Operacje na plikach

```
5 // Zadanie 1.2: Operacje na plikach
6
7
8 List<String> ocenyIZapis = listaOcen
9     .stream()
0     .map(ocena -> "Ocena: " + ocena)
1     .collect(Collectors.toList());
2
3 Collections.sort(ocenyIZapis);
4
5 try {
6     BufferedWriter writer = new BufferedWriter(new FileWriter("oceny.txt"));
7     for (String ocena : ocenyIZapis) {
8         writer.write(ocena);
9         writer.newLine();
0     }
1     writer.close();
2 } catch (IOException e) {
3     e.printStackTrace();
4 }
5
```

### Zadanie 1.3. Operacje na plikach

```
// Zadanie 1.3: Operacje na plikach

List<String> ocenyIZPliku = new ArrayList<>();

try {
    BufferedReader reader = new BufferedReader(new FileReader("oceny.txt"));
    String linia;
    while ((linia = reader.readLine()) != null) {
        ocenyIZPliku.add(linia);
    }
    reader.close();
} catch (IOException e) {
    e.printStackTrace();
}

double srednia = ocenyIZPliku.stream()
    .mapToDouble(ocena -> Integer.parseInt(ocena.substring(7)))
    .average()
    .orElse(0);

Optional<String> najlepszaOcena = ocenyIZPliku.stream()
    .max(Comparator.comparingInt(ocena -> Integer.parseInt(ocena.substring(7))));
Optional<String> najgorszaOcena = ocenyIZPliku.stream()
    .min(Comparator.comparingInt(ocena -> Integer.parseInt(ocena.substring(7))));

System.out.println("Średnia ocen: " + srednia);
System.out.println("Najlepsza ocena: " + najlepszaOcena.orElse("Brak ocen"));
System.out.println("Najgorsza ocena: " + najgorszaOcena.orElse("Brak ocen"));
}
```

Wyniki działania kodu:

```
"C:\Program Files\Java\jdk-17\bin
1.1: Ocena: 4
1.1: Ocena: 3
1.1: Ocena: 5
1.1: Ocena: 4
Ocena 4 powtarza się 2 razy.
Średnia ocen: 4.0
Najlepsza ocena: Ocena: 5
Najgorsza ocena: Ocena: 3
```

```
oceny — Notatnik
Plik  Edycja  Format  Widok  F
Ocena: 3
Ocena: 4
Ocena: 4
Ocena: 5
```

Wnioski:

Wykorzystanie strumieni w Javie pozwala na wygodne i efektywne przetwarzanie danych, zarówno w przypadku operacji na kolekcjach, jak i operacji na plikach. Funkcje takie jak filter, map i collect są użytecznymi narzędziami do manipulacji danymi w strumieniach, umożliwiając filtrowanie, transformację i zbieranie wyników w odpowiednich strukturach danych.

## LABORATORIUM 2. WYKORZYSTANIE PLIKÓW XML.

### Zadanie 2.1. Tworzenie XML

```
1 package com.company;
2 public class Student {
3     private String name;
4     private int age;
5     private String course;
6
7     public Student(String name, int age, String course) {
8         this.name = name;
9         this.age = age;
10        this.course = course;
11    }
12
13    public String getName() {
14        return name;
15    }
16
17    public void setName(String name) {
18        this.name = name;
19    }
20
21    public int getAge() {
22        return age;
23    }
24
25    public void setAge(int age) {
26        this.age = age;
27    }
28
29    public String getCourse() {
30        return course;
31    }
32
33    public void setCourse(String course) {
34        this.course = course;
35    }
36 }
```

```
public class Main {  
  
    public static void main(String[] args) {  
        List<Student> students = new ArrayList<>();  
        students.add(new Student( name: "John Doe", age: 20, course: "Computer Science"));  
        students.add(new Student( name: "Jane Smith", age: -3, course: "Mathematics"));  
        students.add(new Student( name: "Bob Johnson", age: 22, course: "Physics"));  
  
        createXML(students, filename: "students.xml");  
        List<Student> parsedStudents = parseXML( filename: "students.xml");  
  
        for (Student student : parsedStudents) {  
            if (student != null) {  
                System.out.println("Name: " + student.getName());  
                System.out.println("Age: " + student.getAge());  
                System.out.println("Course: " + student.getCourse());  
                System.out.println();  
            } else {  
                System.out.println("Nie można utworzyć obiektu Student");  
            }  
        }  
    }  
}
```



```

public static void createXML(List<Student> students, String filename) {
    try {
        DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

        Document doc = docBuilder.newDocument();
        Element rootElement = doc.createElement( tagName: "Students");
        doc.appendChild(rootElement);

        for (Student student : students) {
            Element studentElement = doc.createElement( tagName: "Student");
            rootElement.appendChild(studentElement);

            Element nameElement = doc.createElement( tagName: "Name");
            nameElement.appendChild(doc.createTextNode(student.getName()));
            studentElement.appendChild(nameElement);

            Element ageElement = doc.createElement( tagName: "Age");
            ageElement.appendChild(doc.createTextNode(String.valueOf(student.getAge())));
            studentElement.appendChild(ageElement);

            Element courseElement = doc.createElement( tagName: "Course");
            courseElement.appendChild(doc.createTextNode(student.getCourse()));
            studentElement.appendChild(courseElement);
        }

        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();
        DOMSource source = new DOMSource(doc);
        StreamResult result = new StreamResult(new File(filename));
        transformer.transform(source, result);

        System.out.println("XML file saved as " + filename);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

## Zadanie 2.2. Odczyt XML

```
public class Main {  
  
    public static void main(String[] args) {  
        List<Student> students = new ArrayList<>();  
        students.add(new Student( name: "John Doe", age: 20, course: "Computer Science"));  
        students.add(new Student( name: "Jane Smith", age: -3, course: "Mathematics"));  
        students.add(new Student( name: "Bob Johnson", age: 22, course: "Physics"));  
  
        createXML(students, filename: "students.xml");  
        List<Student> parsedStudents = parseXML( filename: "students.xml");  
  
        for (Student student : parsedStudents) {  
            if (student != null) {  
                System.out.println("Name: " + student.getName());  
                System.out.println("Age: " + student.getAge());  
                System.out.println("Course: " + student.getCourse());  
                System.out.println();  
            } else {  
                System.out.println("Nie można utworzyć obiektu Student");  
            }  
        }  
    }  
}
```

```

public static List<Student> parseXML(String filename) {
    List<Student> students = new ArrayList<>();

    try {
        File xmlFile = new File(filename);
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(xmlFile);

        doc.getDocumentElement().normalize();

        NodeList studentNodes = doc.getElementsByTagName("Student");

        for (int i = 0; i < studentNodes.getLength(); i++) {
            Node studentNode = studentNodes.item(i);
            if (studentNode.getNodeType() == Node.ELEMENT_NODE) {
                Element studentElement = (Element) studentNode;
                String name = studentElement.getElementsByTagName("Name").item(0).getTextContent();
                String ageStr = studentElement.getElementsByTagName("Age").item(0).getTextContent();
                String course = studentElement.getElementsByTagName("Course").item(0).getTextContent();
                int age;

                try {
                    age = Integer.parseInt(ageStr);
                } catch (NumberFormatException e) {
                    age = -1;
                }

                if (!name.isEmpty() && age > 0 && !course.isEmpty()) {
                    students.add(new Student(name, age, course));
                } else {
                    students.add(null);
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return students;
}

```

### Zadanie 2.3. Filtracja XML

```

public static List<Student> parseXML(String filename) {
    List<Student> students = new ArrayList<>();

    try {
        File xmlFile = new File(filename);
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(xmlFile);

        doc.getDocumentElement().normalize();

        NodeList studentNodes = doc.getElementsByTagName("Student");

        for (int i = 0; i < studentNodes.getLength(); i++) {
            Node studentNode = studentNodes.item(i);
            if (studentNode.getNodeType() == Node.ELEMENT_NODE) {
                Element studentElement = (Element) studentNode;
                String name = studentElement.getElementsByTagName("Name").item(0).getTextContent();
                String ageStr = studentElement.getElementsByTagName("Age").item(0).getTextContent();
                String course = studentElement.getElementsByTagName("Course").item(0).getTextContent();
                int age;

                try {
                    age = Integer.parseInt(ageStr);
                } catch (NumberFormatException e) {
                    age = -1;
                }

                if (!name.isEmpty() && age > 0 && !course.isEmpty()) {
                    students.add(new Student(name, age, course));
                } else {
                    students.add(null);
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return students;
}

```

```

for (Student student : parsedStudents) {
    if (student != null) {
        System.out.println("Name: " + student.getName());
        System.out.println("Age: " + student.getAge());
        System.out.println("Course: " + student.getCourse());
        System.out.println();
    } else {
        System.out.println("Nie można utworzyć obiektu Student");
    }
}
}

```


```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <Students>
3   <Student>
4     <Name>John Doe</Name>
5     <Age>20</Age>
6     <Course>Computer Science</Course>
7   </Student>
8   <Student>
9     <Name>Bob Johnson</Name>
10    <Age>22</Age>
11    <Course>Physics</Course>
12  </Student>
13  <Student>
14    <Name>Jane Smith</Name>
15    <Age>-3</Age>
16    <Course>Mathematics</Course>
17  </Student>
18 </Students>
```

## Wnioski:

Praca z danymi w formacie XML jest powszechna w dzisiejszych aplikacjach. Pozwala na przechowywanie, wymianę i analizę danych w sposób strukturalny. Mechanizmy filtrowania pozwalają na sprawdzenie czy dane spełniają określone kryteria, co jest istotne w przypadku danych, które nie zawsze są kompletnie zgodne lub poprawne.

## LABORATORIUM 6. TWORZENIE APLIKACJI Z WYKORZYSTANIEM BIBLIOTEKI LOMBOK.

Zadanie 6.1. Klasa modelowa – bez użycia biblioteki Lombok



```
1 package com.company;
2
3 public class Employee {
4     private String name;
5     private Integer salary;
6     private Integer age;
7     private String role;
8
9     public Employee(String name, Integer salary, Integer age, String role) {
10         this.name = name;
11         this.salary = salary;
12         this.age = age;
13         this.role = role;
14     }
15
16     public Employee() {
17     }
18
19     public String getName() {
20         return name;
21     }
22
23     public void setName(String name) {
24         this.name = name;
25     }
26
27     public Integer getSalary() {
28         return salary;
29     }
30
31     public void setSalary(Integer salary) {
32         this.salary = salary;
33     }
34
35     public Integer getAge() {
36         return age;
37     }
38
39     public void setAge(Integer age) {
40         this.age = age;
41     }
42 }
```

```
Main.java x Employee.java x
1 package com.company;
2
3 public class Main {
4
5     public static void main(String[] args) {
6
7         Employee employee1 = new Employee( name: "John", salary: 50000, age: 30, role: "Manager");
8         Employee employee2 = new Employee( name: "Jane", salary: 60000, age: 35, role: "CTO");
9         Employee employee3 = new Employee( name: "Jack", salary: 40000, age: 25, role: "Developer");
10
11
12
13     }
```

```
Main.java x Employee.java x
package com.company;

3 related problems
public class Employee {
    private String name;
    private Integer salary;
    private Integer age;
    private String role;
    private Integer DaysOffWork;

    3 related problems
    public Employee(String name, Integer salary, Integer age, String role, Integer daysOffWork) {
        this.name = name;
        this.salary = salary;
        this.age = age;
        this.role = role;
        DaysOffWork = daysOffWork;
    }

    3 related problems
    public Employee() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getSalary() {
        return salary;
    }

    public void setSalary(Integer salary) {
        this.salary = salary;
    }

    public Integer getAge() {
        return age;
    }
}
```

Terminal

## Zadanie 6.2. Klasa modelowa – z biblioteką Lombok

```
workspace.xml x runConfigurations.xml x project-template.xml  
package com.company;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter @Setter @NoArgsConstructor @AllArgsConstructor  
public class Employee {  
    private String name;  
    private Integer salary;  
    private Integer age;  
    private String role;  
    private Integer DaysOffWork;  
}
```

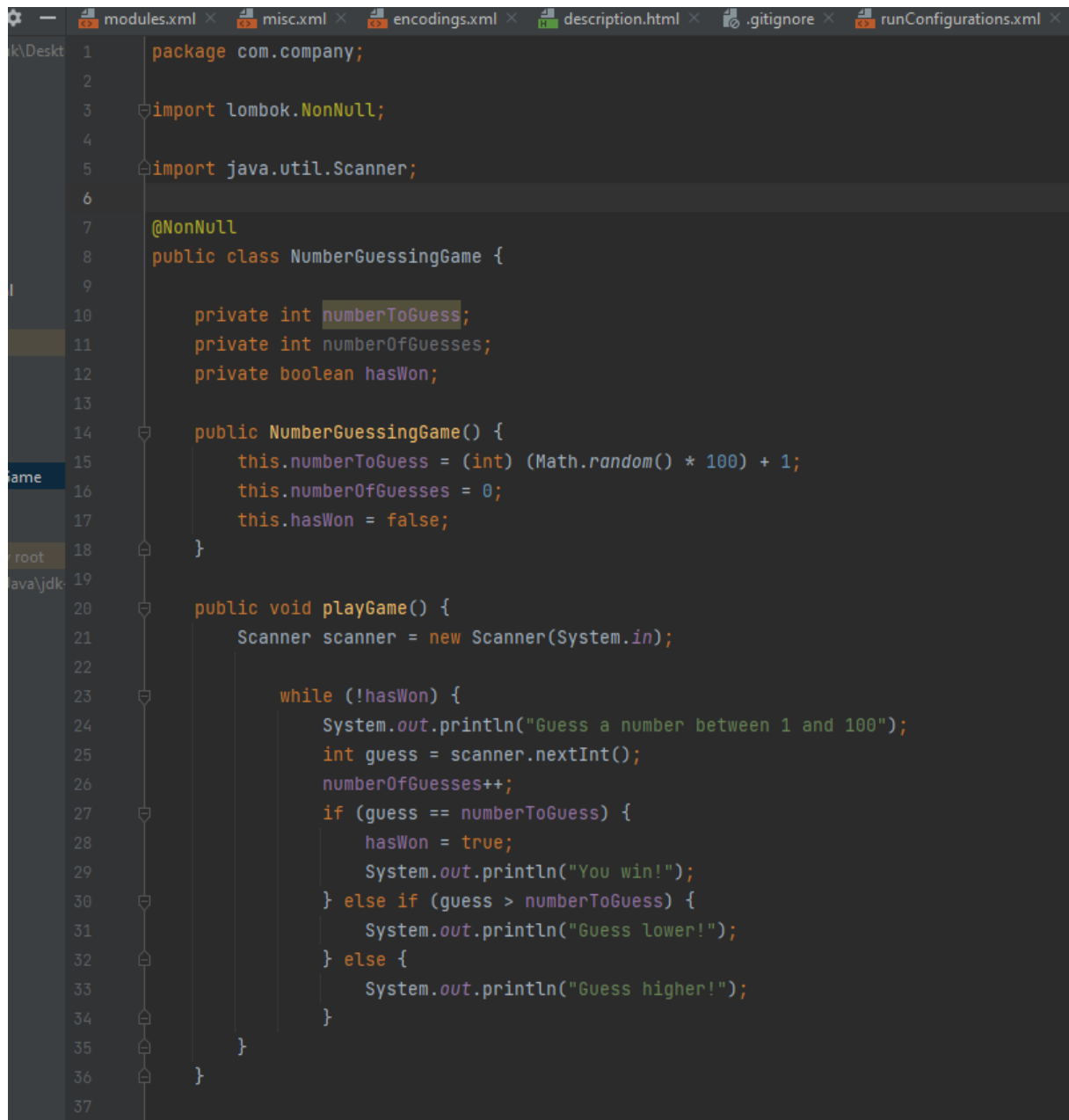
```
workspace.xml x runConfigurations.xml x project-template.xml x E  
package com.company;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Getter @Setter @NoArgsConstructor @AllArgsConstructor  
public class Employee {  
    private String name;  
    private Integer salary;  
    private Integer age;  
    private String role;  
  
}
```



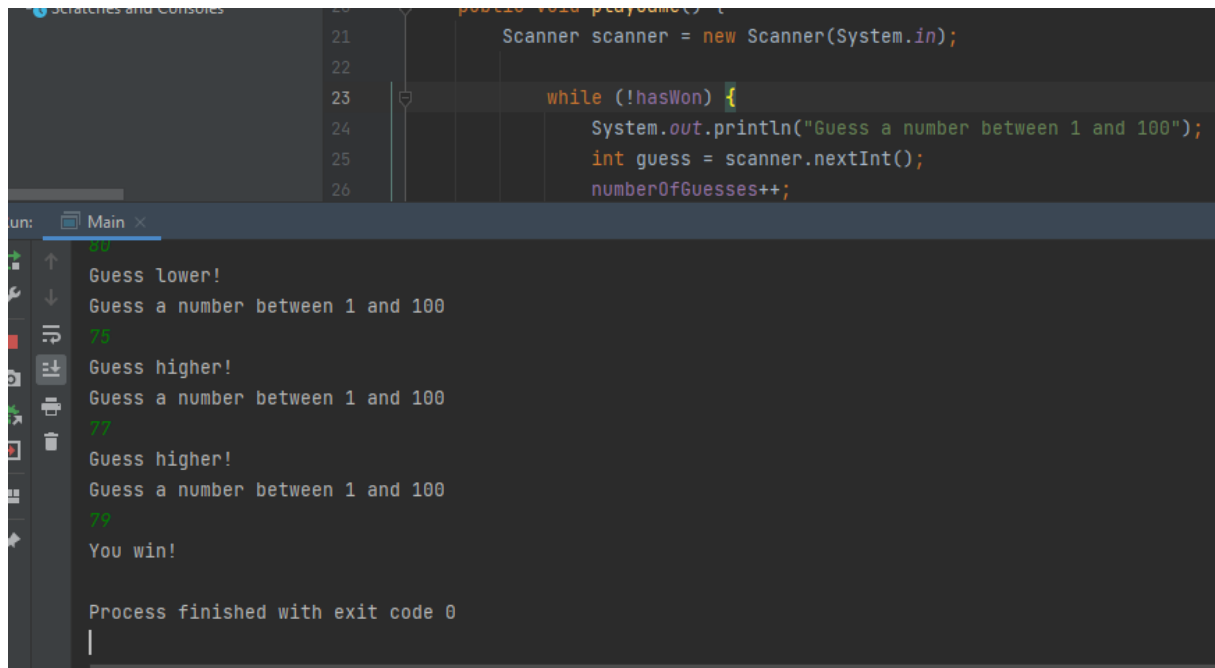
Czy po zmianie liczby pól w klasie modelowej należy coś zmieniać?

Nic nie trzeba zmieniać, Lombok sam aktualizuje na bieżąco gettery, settery oraz konstruktory.

### Zadanie 6.3. Dane od użytkownika



```
1 package com.company;
2
3 import lombok.NonNull;
4
5 import java.util.Scanner;
6
7 @NonNull
8 public class NumberGuessingGame {
9
10     private int numberToGuess;
11     private int numberOfGuesses;
12     private boolean hasWon;
13
14     public NumberGuessingGame() {
15         this.numberToGuess = (int) (Math.random() * 100) + 1;
16         this.numberOfGuesses = 0;
17         this.hasWon = false;
18     }
19
20     public void playGame() {
21         Scanner scanner = new Scanner(System.in);
22
23         while (!hasWon) {
24             System.out.println("Guess a number between 1 and 100");
25             int guess = scanner.nextInt();
26             numberOfGuesses++;
27             if (guess == numberToGuess) {
28                 hasWon = true;
29                 System.out.println("You win!");
30             } else if (guess > numberToGuess) {
31                 System.out.println("Guess lower!");
32             } else {
33                 System.out.println("Guess higher!");
34             }
35         }
36     }
37 }
```



The screenshot shows an IDE with two panels. The top panel displays Java code for a number guessing game. The code includes a Scanner for input, a while loop for the game logic, and System.out.println for output. The bottom panel shows the console output of the program, which includes prompts for guesses, feedback messages like 'Guess lower!' and 'Guess higher!', and a final 'You win!' message. The console also shows the process finished with exit code 0.

```
21 Scanner scanner = new Scanner(System.in);
22
23 while (!hasWon) {
24     System.out.println("Guess a number between 1 and 100");
25     int guess = scanner.nextInt();
26     numberOfGuesses++;
27 }
```

Run: Main x

60  
Guess lower!  
Guess a number between 1 and 100  
75  
Guess higher!  
Guess a number between 1 and 100  
77  
Guess higher!  
Guess a number between 1 and 100  
79  
You win!  
  
Process finished with exit code 0

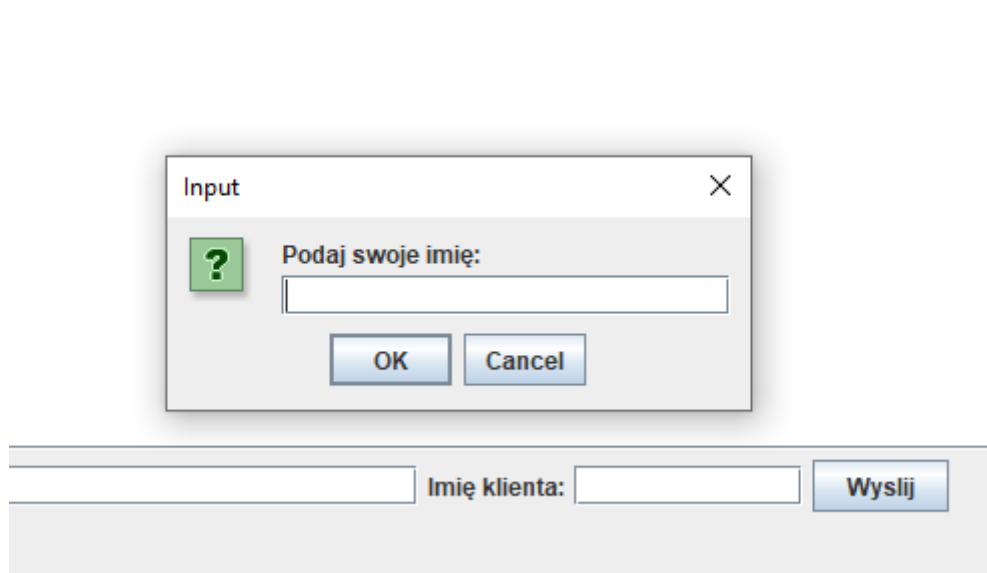
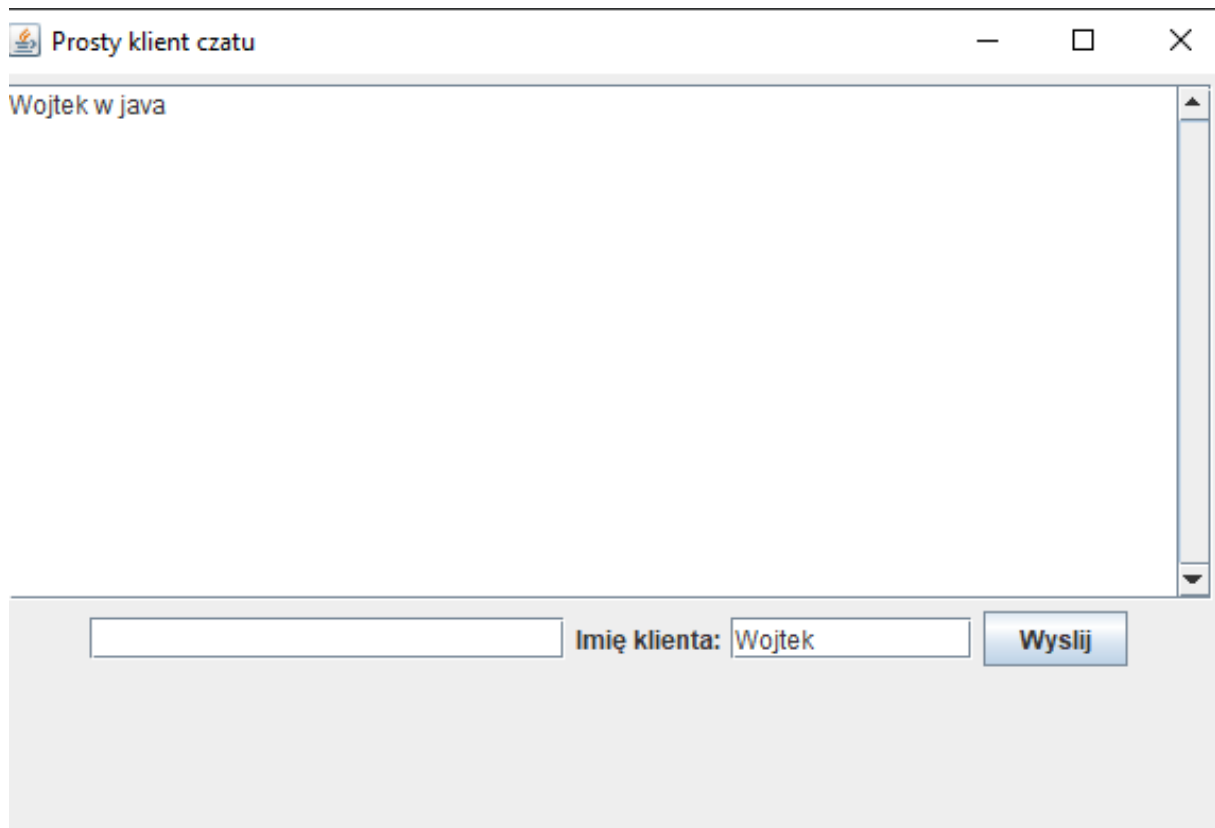
## Wnioski:

Lombok znacznie przyspiesza nam zmiany dokonywane w klasach modelowych, automatycznie gdy zmieniamy pola generuje nam odpowiednie metody. Dodatkowo znacznie skraca ilość kodu oraz zapewnia większą czytelność.

## LABORATORIUM 3. TWORZENIE APLIKACJI SIECIOWYCH.

### Zadanie 3.1. Komunikator sieciowy – GUI

### Zadanie 3.2. Komunikator sieciowy - szyfr



Input

Podaj klucz Vigenere'a:

OK Cancel

Imię klienta:

Wyslij

Nawiązano połączenie!  
Odczytano: ;¥Q@S@

Zakończono konfigurację sieci  
Tu  
Odczytano zaszyfrowaną wiadomość: ;¥Q@S@  
Odczytano zaszyfrowaną wiadomość: ;¥Q@S@  
Klucz Vigenere'a: 123  
Odszyfrowana wiadomość: Wojtek w java  
|

```

package com.example.lab3;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Iterator;

public class Serwer {
    ArrayList<String> zaszyfrowaneWiadomosci;
    ArrayList strumienieWyjsciowe;

    public Serwer() {
        zaszyfrowaneWiadomosci = new ArrayList<>();
        strumienieWyjsciowe = new ArrayList();
    }

    public class ObslugaKlienta implements Runnable {
        BufferedReader czytelnik;
        Socket gniazdo;

        public ObslugaKlienta(Socket gniazdo) {
            try {
                this.gniazdo = gniazdo;
                InputStreamReader reader = new InputStreamReader(gniazdo.getInputStream());
                czytelnik = new BufferedReader(reader);
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }

        @Override
        public void run() {
            String zaszyfrowanaWiadomosc;
            try {
                while ((zaszyfrowanaWiadomosc = czytelnik.readLine()) != null) {
                    System.out.println("Odczytano: " + zaszyfrowanaWiadomosc);
                    zaszyfrowaneWiadomosci.add(zaszyfrowanaWiadomosc);
                    rozeslijDoWszystkich(zaszyfrowanaWiadomosc);
                }
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
}

```

```

41     }
42 }
43 }
44 public static void main(String[] args) {
45     new Serwer().doRoboty();
46 }
47 public void doRoboty() {
48     strumienieWyjsciowe = new ArrayList();
49     try {
50         ServerSocket gniazdoSerwera = new ServerSocket( port: 2021);
51         while (true) {
52             Socket gniazdoKlienta = gniazdoSerwera.accept();
53             PrintWriter pisarz = new PrintWriter(gniazdoKlienta.getOutputStream());
54             strumienieWyjsciowe.add(pisarz);
55             Thread watekKlienta = new Thread(new ObslugaKlienta(gniazdoKlienta));
56             watekKlienta.start();
57             System.out.println("Nawiazano połączenie!");
58         }
59     } catch (Exception ex) {
60         ex.printStackTrace();
61     }
62 }
63 public void rozeslijDoWszystkich(String wiadomosc) {
64     Iterator it = strumienieWyjsciowe.iterator();
65     while (it.hasNext()) {
66         try {
67             PrintWriter pisarz = (PrintWriter) it.next();
68             pisarz.println(wiadomosc);
69             pisarz.flush();
70         } catch (Exception ex) {
71             ex.printStackTrace();
72         }
73     }
}
}
}

```

```

1  package com.example.lab3;
2
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.io.*;
6  import java.net.*;
7  import javax.swing.*;
8
9  public class Klient {
10     JTextArea odbiorWiadomosci;
11     JTextField wiadomosc;
12     JTextField imieField;
13     String imieKlienta;
14     BufferedReader czytelnik;
15     PrintWriter pisarz;
16     Socket gniazdo;
17     String kluczVigenere;
18     public static void main(String[] args) {
19         Klient klient = new Klient();
20         klient.polaczMnie();
21     }
22     public void polaczMnie() {
23         JFrame frame = new JFrame( title: "Prosty klient czatu");
24         JPanel panel = new JPanel();
25         odbiorWiadomosci = new JTextArea( rows: 15, columns: 50);
26         odbiorWiadomosci.setLineWrap(true);
27         odbiorWiadomosci.setWrapStyleWord(true);
28         odbiorWiadomosci.setEditable(false);
29         JScrollPane przewijanie = new JScrollPane(odbiorWiadomosci);
30         przewijanie.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
31         przewijanie.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
32         wiadomosc = new JTextField( columns: 20);
33         imieField = new JTextField( columns: 10);
34         JButton przyciskWyslij = new JButton( text: "Wyslij");
35         przyciskWyslij.addActionListener(new SluchaczPrzycisku());
36
37         JLabel imieLabel = new JLabel( text: "Imię klienta:");
38
39         panel.add(przewijanie);
40         panel.add(wiadomosc);
41         panel.add(imieLabel);
42         panel.add(imieField);

```

```

41     panel.add(imieLabel);
42     panel.add(imieField);
43     panel.add(przyciskWyslij);
44
45     konfiguruj();
46     Thread watekOdbiorcy = new Thread(new Odbiorca());
47     watekOdbiorcy.start();
48
49     frame.getContentPane().add(BorderLayout.CENTER, panel);
50     frame.setSize(new Dimension( width: 600, height: 400));
51     frame.setVisible(true);
52
53     imieKlienta = JOptionPane.showInputDialog(frame, message: "Podaj swoje imię:");
54     kluczVigenere = JOptionPane.showInputDialog(frame, message: "Podaj klucz Vigenere'a:");
55 }
56 private void konfiguruj() {
57     try {
58         gniazdo = new Socket( host: "127.0.0.1", port: 2021);
59         InputStreamReader czytelnikStrm = new InputStreamReader(gniazdo.getInputStream());
60         czytelnik = new BufferedReader(czytelnikStrm);
61         pisarz = new PrintWriter(gniazdo.getOutputStream());
62         System.out.println("Zakończono konfigurację sieci");
63     } catch (IOException ex) {
64         System.out.println("Konfiguracja sieci nie powiodła się!");
65         ex.printStackTrace();
66     }
67 }
68 @ private String zaszyfrujWiadomosc(String wiadomosc) {
69     StringBuilder zaszyfrowanaWiadomosc = new StringBuilder();
70     int wiadomoscLength = wiadomosc.length();
71     int kluczLength = kluczVigenere.length();
72
73     for (int i = 0; i < wiadomoscLength; i++) {
74         char znak = wiadomosc.charAt(i);
75         char kluczZnak = kluczVigenere.charAt(i % kluczLength);
76         char zaszyfrowanyZnak = (char) (znak + kluczZnak);
77         zaszyfrowanaWiadomosc.append(zaszyfrowanyZnak);
78     }
79
80     return zaszyfrowanaWiadomosc.toString();
81 }

```



```

82
83 @ private String deszyfrujWiadomosc(String zaszyfrowanaWiadomosc) {
84     StringBuilder odszyfrowanaWiadomosc = new StringBuilder();
85     int wiadomoscLength = zaszyfrowanaWiadomosc.length();
86     System.out.println("Odczytano zaszyfrowaną wiadomość: " + zaszyfrowanaWiadomosc);
87     System.out.println("Klucz Vigenere'a: " + kluczVigenere);
88
89     int kluczLength = kluczVigenere.length();
90
91     for (int i = 0; i < wiadomoscLength; i++) {
92         char znak = zaszyfrowanaWiadomosc.charAt(i);
93         char kluczZnak = kluczVigenere.charAt(i % kluczLength);
94         char odszyfrowanyZnak = (char) (znak - kluczZnak);
95         odszyfrowanaWiadomosc.append(odszyfrowanyZnak);
96     }
97
98     return odszyfrowanaWiadomosc.toString();
99 }
100
101
102 @ private String formatujWiadomosc(String wiadomosc) {
103     return imieKlienta + ": " + wiadomosc;
104 }
105
106
107 private class SluchaczPrzycisku implements ActionListener {
108     @Override
109     public void actionPerformed(ActionEvent e) {
110         try {
111             /*String sformatowanaWiadomosc = formatujWiadomosc(wiadomosc.getText());
112             pisarz.println(sformatowanaWiadomosc);*/
113             String zaszyfrowanaWiadomosc = zaszyfrujWiadomosc(wiadomosc.getText());
114             pisarz.println(zaszyfrowanaWiadomosc);
115             pisarz.flush();
116         } catch (Exception ex) {
117             ex.printStackTrace();
118         }
119         wiadomosc.setText("");
120         wiadomosc.requestFocus();
121     }
122 }
123 public class Odbiornica implements Runnable {

```

```

123 public class Odbiorca implements Runnable {
124     @Override
125     public void run() {
126         String zaszyfrowanaWiadomosc;
127         System.out.println("Tu");
128         try {
129             while ((zaszyfrowanaWiadomosc = czytelnik.readLine()) != null) {
130                 System.out.println("Odczytano zaszyfrowaną wiadomość: " + zaszyfrowanaWiadomosc);
131
132                 // Deszyfrowanie wiadomości
133                 String odszyfrowanaWiadomosc = deszyfrujWiadomosc(zaszyfrowanaWiadomosc);
134                 System.out.println("Odszyfrowana wiadomość: " + odszyfrowanaWiadomosc);
135
136                 // Wyświetlenie odszyfrowanej wiadomości na interfejsie
137                 odbiorWiadomosci.append(odszyfrowanaWiadomosc + "\n");
138             }
139         } catch (Exception ex) {
140             ex.printStackTrace();
141         }
142     }
143 }
144 }

```

## Wnioski:

Zadania 3.1 i 3.2 przedstawiają podstawową implementację komunikatora sieciowego z interfejsem graficznym oraz dodają funkcję szyfrowania i deszyfrowania wiadomości, co zwiększa bezpieczeństwo komunikacji. Jednak w rzeczywistym środowisku, z uwagi na bezpieczeństwo, należy uwzględnić bardziej zaawansowane mechanizmy, takie jak uwierzytelnianie i bezpieczeństwo komunikacji.

# LABORATORIUM 10. TWORZENIE APLIKACJI Z WYKORZYSTANIEM BIBLIOTEKI JSOUP.

## Zadanie 10.1. Dostęp do treści

```
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
```

Zakład: Zakład Programowania i Grafiki Komputerowej

- dr Marcin Barszcz
- dr hab. inż. Jerzy Montusiewicz, prof. uczelni
- dr inż. Elżbieta Miłoś
- dr inż. Jacek Kęsik
- dr inż. Kamil Żyła
- dr inż. Krzysztof Dziedzic
- dr inż. Maria Skublewska-Paszkowska
- dr inż. Sylwester Korga
- mgr inż. Stanisław Skulimowski

Zakład: Zakład Podstaw Informatyki i Modelowania Komputerowego

- dr Beata Pańczyk
- dr Edyta Łukasik
- dr hab. Tomasz Zientarski, prof. uczelni
- dr inż. Bogusław Oleksiejuk
- dr inż. Dariusz Gutek
- dr inż. Jakub Smółka
- dr inż. Maciej Pańczyk
- mgr Magdalena Zoła
- mgr inż. Kinga Chwaleba
- mgr inż. Weronika Wach

Zakład: Zakład Inżynierii Oprogramowania i Systemów Baz Danych

- dr Mariusz Dzieńkowski
- dr inż. Marek Miłoś, prof. uczelni
- dr inż. Małgorzata Plechawska-Wójcik
- dr inż. Piotr Muryjas
- mgr inż. Magdalena Chmielewska
- mgr inż. Marta Dziuba-Kozieł

Zakład: Zakład Ochrony Informacji i Systemów Operacyjnych

- dr inż. Grzegorz Kozieł
- dr inż. Marcin Badurowicz
- dr inż. Piotr Kopniak
- dr inż. Sławomir Przyłucki
- dr inż. Tomasz Szymczyk

- dr inż. Sławomir Przyłocki
- dr inż. Tomasz Szymczyk

Zakład: Zakład Internetu Rzeczy i Sztucznej Inteligencji

- dr Paweł Powroźnik
- dr hab. Małgorzata Charytanowicz, prof. uczelni
- dr inż. Tomasz Nowicki
- mgr inż. Bartosz Sterniczuk
- mgr inż. Katarzyna Baran
- mgr inż. Piotr Wójcicki

Zakład: Pracownia techniczna

- dr Leszek Sałamacha
- mgr Anna Sałamacha

```

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import java.io.IOException;
import java.util.*;

public class PollubScraper {
    public static void main(String[] args) {
        String url = "https://cs.pollub.pl/staff/";

        try {
            Document doc = Jsoup.connect(url).get();

            Element departmentContainer = doc.selectFirst("div.post-content");
            Elements elements = departmentContainer.getAllElements();
            String currentDepartment = null;
            List<String> currentEmployees = new ArrayList<>();

            for (Element element : elements) {
                if (element.tagName().equals("h3")) {
                    String departmentName = element.text();
                    if (!departmentName.contains("Kierownik Katedry")) {
                        if (currentDepartment != null) {
                            // Posortowanie pracowników alfabetycznie
                            Collections.sort(currentEmployees);

                            // Wyświetlenie posortowanych pracowników
                            for (String employee : currentEmployees) {
                                System.out.println("- " + employee);
                            }
                            currentEmployees.clear();
                        }
                        currentDepartment = departmentName;
                        System.out.println("\n" + "Zakład: " + currentDepartment);
                    }
                } else if (currentDepartment != null && element.tagName().equals("a")) {
                    currentEmployees.add(element.text());
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

## Zadanie 10.2. Filtrowanie

```
if (currentDepartment != null) {
    // Posortowanie pracowników alfabetycznie
    Collections.sort(currentEmployees);

    // Wyświetlenie posortowanych pracowników
    for (String employee : currentEmployees) {
        System.out.println("- " + employee);
    }

    // Wywołanie funkcji filterEmployees do filtrowania pracowników
    List<String> filteredEmployees = filterEmployees(currentEmployees);
    currentEmployees.clear();
    if (!filteredEmployees.isEmpty()) {
        System.out.println("\nOsoby z stopniem dr inż.: \n");
        for (String filteredEmployee : filteredEmployees) {
            System.out.println("- " + filteredEmployee);
        }
    }
    currentDepartment = departmentName;
    System.out.println("\nZakład: " + currentDepartment);
} else if (currentDepartment != null && element.tagName().equals("a")) {
    currentEmployees.add(element.text());
}
} catch (IOException e) {
    e.printStackTrace();
}
}

private static List<String> filterEmployees(List<String> employees) {
    List<String> filteredEmployees = new ArrayList<>();
    for (String employee : employees) {
        if (employee.contains("dr inż.")) {
            filteredEmployees.add(employee);
        }
    }
    return filteredEmployees;
}
```

Zakład: Zakład Programowania i Grafiki Komputerowej

- dr Marcin Barszcz
- dr hab. inż. Jerzy Montusiewicz, prof. uczelni
- dr inż. Elżbieta Miłoś
- dr inż. Jacek Kęsik
- dr inż. Kamil Żyła
- dr inż. Krzysztof Dziedzic
- dr inż. Maria Skublewska-Paszkowska
- dr inż. Sylwester Korga
- mgr inż. Stanisław Skulimowski

Osoby z stopniem dr inż.:

- dr inż. Elżbieta Miłoś
- dr inż. Jacek Kęsik
- dr inż. Kamil Żyła
- dr inż. Krzysztof Dziedzic
- dr inż. Maria Skublewska-Paszkowska
- dr inż. Sylwester Korga

Zakład: Zakład Podstaw Informatyki i Modelowania Komputerowego

- dr Beata Pańczyk
- dr Edyta Łukasik
- dr hab. Tomasz Zientarski, prof. uczelni
- dr inż. Bogusław Oleksiejuk
- dr inż. Dariusz Gutek
- dr inż. Jakub Smółka
- dr inż. Maciej Pańczyk
- mgr Magdalena Zoła
- mgr inż. Kinga Chwaleba
- mgr inż. Weronika Wach

Osoby z stopniem dr inż.:

- dr inż. Bogusław Oleksiejuk
- dr inż. Dariusz Gutek
- dr inż. Jakub Smółka
- dr inż. Maciej Pańczyk

### Zadanie 10.3. Wyszukiwanie

```
public class SearchingData {
    public static void main(String[] args) {
        List<Event> events = searchGodzinyDziekanske( url: "https://pollub.pl/wyszukiwarka?query=godziny+dziekańskie");

        // Wyświetlenie wyników
        for (Event event : events) {
            System.out.println(event);
        }
    }

    public static List<Event> searchGodzinyDziekanske(String url) {
        List<Event> events = new ArrayList<>();

        try {
            Document doc = Jsoup.connect(url).get();
            Elements results = doc.select( cssQuery: "h4");

            for (Element result : results) {
                String title = result.text();
                //System.out.println(title);
                String dateStr = result.select( cssQuery: "small").text();
                //System.out.println(dateStr);
                if (!dateStr.isEmpty()) {
                    SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "yyyy-MM-dd");
                    Date date = dateFormat.parse(dateStr);
                    events.add(new Event(title, date));
                }
            }

            // Sortowanie listy zdarzeń względem daty (od najstarszego do najmłodszego)
            events.sort((event1, event2) -> event1.getDate().compareTo(event2.getDate()));
        } catch (IOException | ParseException e) {
            e.printStackTrace();
        }

        return events;
    }
}
```



```

import java.text.SimpleDateFormat;
import java.util.Date;

public class Event {
    private String title;
    private Date date;

    public Event(String title, Date date) {
        this.title = title;
        this.date = date;
    }

    public String getTitle() {
        return title;
    }

    public Date getDate() {
        return date;
    }

    @Override
    public String toString() {
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

        return "Tytuł: " + title + ", Data: " + dateFormat.format(date);
    }
}

```

## Wnioski:

W ramach tych trzech zadań stworzono kompleksową aplikację, która umożliwia dostęp, analizę i przetwarzanie treści ze stron internetowych. Aplikacja pozwala na pobieranie danych, filtrowanie ich oraz przeszukiwanie informacji, zapewniając użytkownikowi spersonalizowany dostęp do konkretnych treści. To doskonały przykład pozyskiwania, sortowania i prezentowania danych w sposób bardziej użyteczny i czytelny.